

LIS User's Guide

Submitted under Task Agreement GSFC-CT-2

Cooperative Agreement Notice (CAN) CAN-00OES-01

Increasing Interoperability and Performance of
Grand Challenge Applications in the Earth, Space, Life, and Microgravity
Sciences

December 2003

Revision 2.3

History:

Revision	Summary of Changes	Date
2.3	Improvements to Milestone "T"	November 30, 2003
2.1	Milestone "T" release	November 10, 2003
2.0	Milestone "T" submission	August 14, 2003
1.1	Milestone "F" release	April 25, 2003
1.0	Milestone "F" submission	March 31, 2003



Goddard Space Flight Center
Greenbelt, Maryland 20771
December 2003

Contents

1	Introduction	4
2	Background	5
2.1	LIS	5
2.2	LIS driver	5
2.3	Community Land Model (CLM)	6
2.4	The Community Noah Land Surface Model	7
2.5	Variable Infiltration Capacity (VIC) Model	7
2.6	GrADS-DODS Server	8
3	Preliminaries	9
4	Running Modes	10
5	Obtaining the Source Code	11
5.1	Downloading the Source Code	11
5.2	Source files	11
5.3	Scripts	13
5.4	Post-processing	13
6	Obtaining the Datasets	14
6.1	Downloading the Source Code	14
6.2	Downloading Parameter Datasets	14
6.3	Downloading the Forcing Datasets	14
7	Building the Executable	15
7.1	General Build Instructions	15
7.1.1	Required Software Libraries	15
7.1.2	Modifying the Makefile	15
7.2	Defining source directories for compilation	16
7.3	Defining components while building the executable	17
7.4	Compiling GrADS-DODS Support	18
7.5	Generating documentation	19
8	Running the Executable	20
8.1	Configuring Run Via LIS Card File	20
8.1.1	driver namelist	20
8.1.2	lis_run_inputs namelist	21
8.1.3	domain namelist	25
8.1.4	parms namelist	27
8.1.5	geos namelist	28
8.1.6	gdas namelist	28
8.1.7	cmap namelist	28
8.1.8	agrmet namelist	28
8.1.9	clm2 namelist	28

8.1.10	noah namelist	28
8.1.11	vic namelist	29
9	Output Data Processing	30
9.1	Building mapto2D	30
9.2	Running mapto2D	31
9.3	CLM Output	32
9.4	Noah Output	34
9.5	VIC Output	36
A	LIS Card File	38
B	Makefile	42

1 Introduction

This is the LIS' User's Guide. This document describes how to download and install the code and data needed to run the LIS executable for LIS' "Interoperability Prototype" milestone – Milestone "I". It describes how to build and run the code, and finally this document also describes how to download output data sets to use for validation. Updates to this document will provide more detailed instructions on how to configure the executable and will address the graphical user interface.

This document consists of 8 sections, described as follows:

- 1 Introduction:** the section you are currently reading
- 2 Background:** general information about the LIS project
- 3 Preliminaries:** general info., steps, instructions, and definitions used throughout the rest of this document
- 4 Running modes:** different parallel running modes of operation
- 5 Obtaining the Source Code:** the steps needed to download the source code.
- 6 Obtaining the Datasets:** the steps needed to download the data sets.
- 7 Building the Executable:** the steps needed to build the LIS executable
- 8 Running the Executable:** the steps needed to prepare and submit a run, also describes the various run-time configurations
- 9 Output Data Processing:** the steps needed to post-process generated output for visualization

2 Background

This section provides some general information about the LIS project and land surface modeling.

2.1 LIS

The primary goal of the LIS project is to build a system that is capable of performing high resolution land surface modeling at high performance using scalable computing technologies. The LIS software system consists of a number of components: (1) LIS driver: the core software that integrates the use of land surface models, data management techniques, and high performance computing. (2) community land surface models such as CLM [3], Noah [5], and VIC [6], and (3) Visualization and data management tools such as GrADS [1] -DODS [4] server. One of the important design goals of LIS is to develop an interoperable system to interface and interoperate with land surface modeling community and other earth system models. LIS is designed using an object oriented, component-based style. The adaptable interfaces in LIS can be used by the developers to ease the cost of development and foster rapid prototyping and development of applications. The following sections describe the main components of LIS.

2.2 LIS driver

The core of LIS software system is the LIS driver that controls program execution. The LIS driver is a model control and input/output system (consisting of a number of subroutines, modules written in Fortran 90 source code) that drives multiple offline one-dimensional LSMs. The one-dimensional LSMs such as CLM and Noah, apply the governing equations of the physical processes of the soil-vegetation-snowpack medium. These land surface models aim to characterize the transfer of mass, energy, and momentum between a vegetated surface and the atmosphere. When there are multiple vegetation types inside a grid box, the grid box is further divided into "tiles", with each tile representing a specific vegetation type within the grid box, in order to simulate sub-grid scale variability.

The execution of the LIS driver starts with reading in the user specifications, including the modeling domain, spatial resolution, duration of the run, etc. Section 8 describes the exhaustive list of parameters specified by the user. This is followed by the reading and computing of model parameters. The time loop begins and forcing data is read, time/space interpolation is computed and modified as necessary. Forcing data is used to specify the boundary conditions to the land surface model. The LIS driver applies time/space interpolation to convert the forcing data to the appropriate resolution required by the model. The selected model is run for a vector of "tiles" and output and restart files are written at the specified output interval.

Some of the salient features provided by the LIS driver include:

- Vegetation type-based “tile” or “patch” approach to simulate sub-grid scale variability.
- Makes use of various satellite and ground-based observational systems.
- Derives model parameters from existing topography, vegetation, and soil coverages.
- Extensible interfaces to facilitate incorporation of new land surface models, forcing schemes.
- Uses a modular, object oriented style design that allows “plug and play” of different features by allowing user to select only the components of interest while building the executable.
- Ability to perform regional modeling (only on the domain of interest).
- Provides a number of scalable parallel processing modes of operation.

Please refer to the software design document for a detailed description of the design of LIS driver. The LIS developer’s guide describes how to use the extensible interfaces in LIS. The “plug and play” feature of different components is described in this document.

2.3 Community Land Model (CLM)

CLM (Community Land Model) is a 1-D land surface model, written in Fortran 90, developed by a grass-roots collaboration of scientists who have an interest in making a general land model available for public use. LIS currently uses CLM version 2.0. CLM version 2.0 was released in May 2002. The source code for CLM 2.0 is freely available from the National Center for Atmospheric Research (NCAR) [3]. The CLM is used as the land model for the Community Climate System Model (CCSM) (<http://www.ccsm.ucar.edu/>), which includes the Community Atmosphere Model (CAM) (<http://www.cgd.ucar.edu/cms/>). CLM is executed with all forcing, parameters, dimensioning, output routines, and coupling performed by an external driver of the user’s design (in this case done by LDAS). CLM requires pre-processed data such as the land surface type, soil and vegetation parameters, model initialization, and atmospheric boundary conditions as input. The model applies finite-difference spatial discretization methods and a fully implicit time-integration scheme to numerically integrate the governing equations. The model subroutines apply the governing equations of the physical processes of the soil-vegetation-snowpack medium, including the surface energy balance equation, Richards’ [11] equation for soil hydraulics, the diffusion equation for soil heat transfer, the energy-mass balance equation for the snowpack, and the Collatz et al. [8] formulation for the conductance of canopy transpiration.

2.4 The Community Noah Land Surface Model

The community Noah Land Surface Model is a stand-alone, uncoupled, 1-D column model freely available at the National Centers for Environmental Prediction (NCEP; [5]). The name is an acronym representing the various developers of the model (N: NCEP; O: Oregon State University, Dept. of Atmospheric Sciences; A: Air Force (both AFWA and AFRL - formerly AFGL, PL); and H: Hydrologic Research Lab - NWS (now Office of Hydrologic Development – OHD)). Noah can be executed in either coupled or uncoupled mode. It has been coupled with the operational NCEP mesoscale Eta model [9] and its companion Eta Data Assimilation System (EDAS) [12], and the NCEP Global Forecast System (GFS) and its companion Global Data Assimilation System (GDAS). When Noah is executed in uncoupled mode, near-surface atmospheric forcing data (e.g., precipitation, radiation, wind speed, temperature, humidity) is required as input. Noah simulates soil moisture (both liquid and frozen), soil temperature, skin temperature, snowpack depth, snowpack water equivalent, canopy water content, and the energy flux and water flux terms of the surface energy balance and surface water balance. The model applies finite-difference spatial discretization methods and a Crank-Nicholson time-integration scheme to numerically integrate the governing equations of the physical processes of the soil vegetation-snowpack medium, including the surface energy balance equation, Richards' [11] equation for soil hydraulics, the diffusion equation for soil heat transfer, the energy-mass balance equation for the snowpack, and the Jarvis [10] equation for the conductance of canopy transpiration.

2.5 Variable Infiltration Capacity (VIC) Model

Variable Infiltration Capacity (VIC) model is a macroscale hydrologic model, written in C, being developed at the University of Washington and Princeton University. The VIC code repository along with the model description and source code documentation is publicly available at the Princeton website [6]. VIC is used in macroscopic land use models such as SEA - BASINS (<http://boto.ocean.washington.edu/seasia/intro.htm>). VIC is a semi-distributed, grid-based hydrological model, which parameterizes the dominant hydrometeorological processes taking place at the land surface - atmospheric interface. The execution of VIC model requires preprocessed data such as precipitation, temperature, meteorological forcing, soil and vegetation parameters, etc. as input. The model uses three soil layers and one vegetation layer with energy and moisture fluxes exchanged between the layers. The VIC model represents surface and subsurface hydrologic processes on a spatially distributed (grid cell) basis. Partitioning grid cell areas to different vegetation classes can approximate sub-grid scale variation in vegetation characteristics. VIC models the processes governing the flux and storage of water and heat in each cell-sized system of vegetation and soil structure. The water balance portion of VIC is based on three concepts: 1) Division of grid-cell into fraction sub-grid vegetation coverages.

- 2) The variable infiltration curve for rainfall/runoff partitioning at the land surface.
- 3) A baseflow/deep soil moisture curve for lateral baseflow.

Water balance calculations are preformed at three soil layers and within a vegetation canopy. An energy balance is calculated at the land surface. A full description of algorithms in VIC can be found in the references listed at the VIC website.

2.6 GrADS-DODS Server

A GrADS-DODS Server is a data server built upon the Grid Analysis and Display System (GrADS) and the Distributed Oceanographic Data System (DODS).

GrADS is an earth science data manipulation and visualization tool under development at the Center for Ocean-Land-Atmosphere Studies (COLA) (<http://grads.iges.org/cola.html>). See <http://grads.iges.org/grads/grads.html> for more detailed information about GrADS.

DODS, also called the Open source Project for a Network Data Access Protocol (OPeNDAP), is a protocol for serving datasets stored in various formats over a network. See <http://www.unidata.ucar.edu/packages/dods/> for more detailed information about DODS.

A GDS may be used to provide the LIS driver with the forcing and input parameter data needed to run an LSM.

A GDS is an optional component of the LIS system. LIS may be run without using a GDS to access the forcing and input parameter datasets. All necessary forcing and input parameter datasets may be stored on locally-accessible hard-disks and read in directly by the LIS driver, provided the computer system has sufficient memory.

The intent of a GDS for the LIS project is to provide the LIS driver with subsets of the forcing and input parameter datasets, so that large-scale, high-resolution domains may be broken-up/parallelized and processed across many compute-nodes of a Beowulf cluster.

3 Preliminaries

This code has been compiled and run on both SGI IRIX64 6.5 systems and Linux PC (Intel/AMD based) systems. These instructions expect that you are using such a system. In particular you need

Software:

- SGI
 - MIPSpro version 7.3.1.1m
 - Message Passing Toolkit, mpt, version 1.5.3.0
 - GNU's make, gmake, version 3.77
- Linux
 - Absoft's Pro Fortran Software Developement Kit, version 8.0
or
Lahey/Fujitsu's Fortran 95 Compiler, release L6.00c
 - GNU's C and C++ compilers, gcc and g++, version 2.96
 - MPICH , version 1.2.4
 - GNU's make, gmake, version 3.77

System Resources:

- 250MB to 32GB of memory
- 2.5GB to 46GB of hard disk space
 - 64MB for source
 - 1.5 GB to 25GB for input data
 - 425MB to 20GB for output data

You need to create a working directory on your system that has sufficient disk space to install and run in. Throughout the rest of this document this directory shall be refered to as *\$WORKING*.

4 Running Modes

The computational and resource requirements increase significantly for global modeling at such high resolutions such as 5km and 1km. The land surface modeling component in LIS is designed to handle these requirements and perform high-performance, parallel simulation of global, regional, and local land surface processes with a number of land surface models.

LIS is designed to operate in a number of high performance running modes to meet the diverse requirements of a distributed memory and shared memory platforms. LIS can operate in two different parallel modes based on the way data is handled by the LIS driver. In the message passing interface (MPI)-based paradigm, a master processor handles data for the entire domain, computes domain decomposition, and subsequently distributes data onto the compute nodes. This paradigm is limited by the amount of memory available to the master processor. On a shared memory platform, a pool of processors can be used to make a large amount of memory available. To handle increased memory requirements and the limited resources available on a distributed memory environment, a GDS-based running mode can be used in LIS. In this mode of operation, the compute nodes retrieve data from a GDS server. This mode of operation is no longer constrained by the lack of a large pool of memory on the master processor.

LIS driver also includes the capability to perform regional modeling in addition to global scales. The domain information can be specified by a user, and the LIS driver handles the subsetting tasks. In the MPI-mode, the subsetting information is derived from a larger domain, whereas in the GDS-mode, the subsetting is carried out by requesting appropriate data from the GDS-server. The details of using these different options are described in the following sections.

5 Obtaining the Source Code

This section describes how to obtain the source code and datasets needed to run the LIS executable.

5.1 Downloading the Source Code

To obtain the source code needed for LIS’ “Interoperability Prototype” revision 2.3:

1. Go to LIS’ “Public Release Home Page”
Go to <http://lis.gsfc.nasa.gov/>
Follow the “Source Codes” link.
Follow the “LIS 2.3 Code Release” link.
2. From LIS’ “Public Release Home Page”
Follow the “LIS 2.3 Source Code and Scripts” link.
3. Download the *source.tar.gz*, *scripts.tar.gz*, and the *postproc.tar.gz* files into your working directory, *\$WORKING/LIS*.
4. Unpack these files. Run (in the order listed):

```
% gzip -dc source.tar.gz | tar xf -
% gzip -dc scripts.tar.gz | tar xf -
% gzip -dc postproc.tar.gz | tar xf -
```

Unpacking the *source.tar.gz* file will also create the input directory tree needed for downloading the input datasets.

5.2 Source files

Unpacking the *source.tar.gz* file will create a *\$WORKING/LIS/src* sub-directory, *\$WORKING/LIS/opendap_scripts*, and *\$WORKING/LIS/utils* sub-directories. The structure of *src* is as follows:

Directory Name	Synopsis
driver	LIS driver routines
lsm-plugin	Modules defining the function table registry of included LSMs
forcing-plugin	Modules defining function table registries of included model forcing, observed radiation, and precipitation forcing products.
baseforcing	Top level directory for base forcing methods
baseforcing/geos	Routines for handling GEOS forcing product
baseforcing/gdas	Routines for handling GDAS forcing product
obsprecips	Top level directory for observed precipitation products
obsprecips/cmap	Routines for handling CMAP precipitation product
obsprecips/huff	Routines for handling HUFFMAN precipitation product
obsprecips/pers	Routines for handling PERSIANN precipitation product
obsrads	Top level directory for observed radiation products
obsrads/agrmet	Routines for handling AGRMET radiation product
iplib	Interpolation routines (Adopted from NCEP's ipolates library)
lib	Libraries needed for linking
make	Makefile and needed headers
lsms/clm2	Top level clm2 land surface model sub-directory
lsms/clm2/biogeochem	Biogeochemistry routines
lsms/clm2/biogeophys	Biogeophysics routines (e.g., surface fluxes)
lsms/clm2/camclm_share	Code shared between the clm2 and cam (e.g., calendar information)
lsms/clm2/csm_share	Code shared by all the geophysical model components of the Community Climate System Model (CCSM). Currently contains code for CCSM message passing orbital calculations and system utilities
lsms/clm2/ecosysdyn	Ecosystem dynamics routines (e.g., leaf and stem area index)
clm2/main	Control (driver) routines
clm2/mksrfdata	Routines for generating surface datasets
clm2/riverroute	River routing (RTM) routines
clm2/utils	Independent utility routines
lsms/noah.2.6	Noah land surface model version 2.6
lsms/noah.2.5.2	Noah land surface model version 2.5.2
lsms/vic	VIC land surface model
tables	Contains the GRIB tables for writing grib output

The `$WORKING/LIS/opendap_scripts` directory contains the scripts used by the program to access data from a GDS server. Run the `links.sh` to generate all the scripts needed by LIS driver. The `$WORKING/LIS/utils` directory contains some helpful utilites including the scripts used by the documentation generator.

Source code documentation may be found on LIS's web-site at
<http://lis.gsfc.nasa.gov/Documentation/LIS2.3/lis/index.html>

5.3 Scripts

The *scripts.tar.gz* file contains a script for compiling and building the executable and a sample card file used for running the LIS executable and for configuring the individual runs. These are described in Section 8.

Unpacking the *scripts.tar.gz* file will place the following files into the *\$WORKING/LIS* sub-directory:

File Name	Synopsis
lis.crd	Sample card file
comp.csh	Compile and build script

5.4 Post-processing

The *postproc.tar.gz* file contains the source and data files needed to build and run the post-processing utility *mapto2D*. Post-processing is described in Section 9.

Unpacking the *postproc.tar.gz* file will create a *\$WORKING/LIS/postproc* sub-directory. The structure of *postproc* is as follows:

File Name	Synopsis
compile.sh	Script to compile the mapto2D executable
gridout.ctl	Sample GrADS descriptor file
MASKS	Directory containing files used for 1d to 2d mapping
getmask.f90	
mapto2D.f90	
resolution_module.f90	
tile2grid.f90	

6 Obtaining the Datasets

This section describes how to obtain the source code and datasets needed to run the LIS executable.

6.1 Downloading the Source Code

To obtain the source code needed for LIS’ “Interoperability Prototype” revision 2.3:

1. Go to LIS’ “Home Page”
Go to <http://lis.gsfc.nasa.gov/>
Follow the “Get LIS Data” link.

The Milestone I link provides links to the land surface parameters and atmospheric forcing data.

6.2 Downloading Parameter Datasets

Land surface models simulate the physical and dynamical processes of the land surface. Driven by external forcing, the spatial and temporal evolution of these processes are intrinsically determined by the physical and dynamical properties, or parameters, of the land surface. Please follow the link to land surface parameters under Milestone I to obtain parameter datasets. It is recommended that the files be organized according to the domain resolution and land surface model type.

6.3 Downloading the Forcing Datasets

As mentioned earlier, the land surface models in LIS are forced by model-derived output and satellite and ground-based observations. The datasets are available at link to atmospheric forcing data under Milestone I data page.

7 Building the Executable

This section describes how to build the source code and create LIS' executable, LIS.

First perform the steps described in Section ??.

If you are building on a Linux pc system, you must edit the *Makefile* located in `$WORKING/LIS/src/make`. Change the definition of `MPI_PREFIX` to the directory where you installed MPICH. Currently `MPI_PREFIX` is set to `/data1/jim/local/mpich-1.2.4-absoft`.

Then

1. Change directory into `$WORKING/LIS/`
2. Run: % `./comp.csh`

See Appendix B to see the Makefile.

7.1 General Build Instructions

This section describes how to build the LIS code on a platform other than those discussed in Section 3.

7.1.1 Required Software Libraries

In order to build the LIS executable, the following libraries must be installed on your system:

- Message Passing Interface (MPI)
 - vendor supplied, or
 - MPICH
 - (<http://www-unix.mcs.anl.gov/mpi/mpich/>)
- Earth System Modeling Framework (ESMF) (<http://www.esmf.ucar.edu/>)
- bacio
- w3lib

To install the MPI libraries, follow the instructions provided at the MPI URL listed above.

7.1.2 Modifying the Makefile

This section lists the variables in the Makefile that must be set by the user before compiling.

Variable	Description
UNAMES	set by call to uname to determine what type of system you are using. Determine which variable (UNAMES or UMACHINE) will contain the needed information and use it.
UMACHINE	set by call to uname to determine what type of system you are using. Determine which variable (UNAMES or UMACHINE) will contain the needed information and use it.
MPI_PREFIX	path to where mpi libraries are installed
LIB_MPI	path to mpi libraries
INC_MPI	path to mpi header files
ESMF_DIR	path to where esmf libraries are installed
LIB_ESMF	path to esmf libraries
MOD_ESMF	path to esmf modules
ESMF_ARCH	system on which esmf libraries were compiled
FC	fortran compiler
CPP	C preprocessor
LIB_DIR	path to where lis libraries are installed
CPPFLAGS	flags for C preprocessor
CFLAGS	flags for C compiler
FFLAGS	flags for Fortran compiler
FOPTS	additional options for compiler and linker
LDFLAGS	flags for linker
NEW_ARCH_HERE	replace this with the type of system you are using, either the result from uname -s or uname -m. E.g., IRIX64 or i686

Note: For linux architectures, the default ESMF_ARCH is set to be linux_absoft.
For lahey architectures, ESMF_ARCH needs to be changed to linux_lf95.

7.2 Defining source directories for compilation

A file called *Filepath* in the \$WORKING/LIS/src/make directory specifies all the source files that will be included during compilation. A sample *Filepath* is shown below.

```
../driver
../lsm-plugin
../forcing-plugin
../iplib
../baseforcing/geos
../baseforcing/gdas
../obsprecips/cmap
../obsrads/agrmet
../lsms/noah.2.6
../lsms/mosaic
../lsms/vic
../lsms/clm2
```

```

./lsms/clm2/main
./lsms/clm2/biogeophys
./lsms/clm2/biogeochem
./lsms/clm2/camclm_share
./lsms/clm2/csm_share
./lsms/clm2/riverroute
./lsms/clm2/ecosysdyn

```

7.3 Defining components while building the executable

As described earlier, the design of LIS allows users to define and include only the components of interest while building the executable. Since Fortran is not a truly object oriented language, this type of runtime polymorphism can only be simulated in software.

The LIS developers guide describes how new land surface models, forcing schemes, etc can be included in LIS. This is achieved by allowing the user to specify the extensible interfaces such as the ones provided in `$WORKING/LIS/src/lsm-plugin` and `$WORKING/LIS/src/forcing-plugin` directories. Once the user specifies the components to be used in these interfaces, the *Filepath* directory can be modified to include only these components. For example, if a user is interested in running only one land surface model (say Noah), using only the GEOS forcing scheme, and no observational forcing products, the *Filepath* directory reduces to:

```

./driver
./lsm-plugin
./forcing-plugin
./iplib
./baseforcing/geos
./lsms/noah.2.6

```

The extensible interfaces need to be defined as follows:

The `lsm_plugin` method in `$WORKING/LIS/src/lsm-plugin/lsm_pluginMod.F90` needs to be defined as:

```

subroutine lsm_plugin
    use noah_varder, only : noah_varder_ini
    external noah_main
    external noah_setup
    external noahrst
    external noah_output
    external noah_f2t
    external noah_writerst
    external noah_dynsetup

    call registerlsmmini(1,noah_varder_ini)
    call registerlsmsetup(1,noah_setup)

```

```

call registerlsmdynsetup(1,noah_dynsetup)
call registerlsmrun(1,noah_main)
call registerlsmrestart(1,noahrst)
call registerlsmoutput(1,noah_output)
call registerlsmf2t(1,noah_f2t)
call registerlsmwrst(1,noah_writerst)
end subroutine lsm_plugin

```

The *baseforcing_plugin* method in *\$WORKING/LIS/src/forcing-plugin/baseforcing_pluginMod.F90* needs to be defined as:

```

subroutine baseforcing_plugin
use geosdomain_module
external getgeos
external time_interp_geos
call registerget(2,getgeos)
call registerdefnat(2,defnatgeos)
call registertimeinterp(2,time_interp_geos)
end subroutine baseforcing_plugin

```

The *precipforcing_plugin* method in *\$WORKING/LIS/src/forcing-plugin/precipforcing_pluginMod.F90* and the *radforcing_plugin* method in *\$WORKING/LIS/src/forcing-plugin/radforcing_pluginMod.F90* can be left empty as follows:

```

subroutine precipforcing_plugin
end subroutine precipforcing_plugin

subroutine radforcing_plugin
end subroutine radforcing_plugin

```

Similarly, different combinations of using the components can be implemented defining the interfaces appropriately and choosing the corresponding source files through the *Filepath* file.

7.4 Compiling GrADS-DODS Support

The above building instructions generate a LIS executable that reads input data off of local disk. To compile an executable that uses a GrADS-DODS server ¹ to retrieve input data files, you must edit the *Makefile*. Find the appropriate FFLAGS definition the *Makefile*. Add -DOPENDAP to the end of the definition. Then follow the above building instructions.

¹This LIS distribution is configured to retrieve data using LIS' GrADS-DODS server.

7.5 Generating documentation

LIS code uses the ProTex documenting system [2]. The documentation in LATEXformat can be produced by typing `gmake doc` in the `$WORKING/LIS/src/make` directory. This command produces documentation, generating all the files in `$WORKING/LIS/doc` directory. These files can be easily converted to pdf or html formats using utilites such as `pdflatex` or `latex2html`.

8 Running the Executable

This section describes how to run the LIS executable. Once the LIS executable is built, a simulation can be performed using the `lis.crd` file. As described earlier, LIS can only be executed through an `mpirun` script. Assuming that MPI is installed correctly, the lis simulation can be carried out by the following command on the `$WORKING/LIS` directory

```
% mpirun -np 1 ./LIS
```

The `-np` flag indicates the number of processors used in the run. On a multi-processor machine, the parallel processing capabilities of LIS can be exploited using this flag.

8.1 Configuring Run Via LIS Card File

This section describes how to configure your LIS run by manually editing the Fortran namelists contained in the `lis.crd` “card file”.

Currently, there are only a handful of options that may be reset using the card file. They are described in the following sub-sections. The remaining options should be left untouched.

If you manually edit the `lis.crd` card file, do not run the LIS executable via the shell scripts described above. These scripts will over-write your newly edited `lis.crd` card file. To run the LIS executable, read an appropriate shell script to see the necessary commands.

See Appendix A to see a sample lis card file.

8.1.1 driver namelist

In the `driver` namelist of the card file consists of the following options

```
LIS%d%DOMAIN  
LIS%m%LSM  
LIS%f%FORCE  
LIS%d%SOIL  
LIS%p%LAI
```

`LIS%d%DOMAIN` specifies the resolution for the run. Acceptable values are:

Value	Description
2	1/4 deg resolution
3	2 × 2.5 deg resolution
4	1 deg resolution
5	1/2 deg resolution
6	5 km resolution

`LIS%m%LSM` specifies the land surface to run. Acceptable values are:

Value	Description
1	Noah
2	CLM
3	VIC

LIS%f%FORCE specifies the forcing data source for the run. Acceptable values are:

Value	Description
1	GDAS
2	GEOS

LIS%d%SOIL specifies the forcing data source for the run. Acceptable values are:

Value	Description
1	Original veg-based
2	Reynolds soils

LIS%p%LAI specifies the forcing data source for the run. Acceptable values are:

Value	Description
1	Original veg-based
2	AVHRR-based LAI
3	MOIS-based LAI

8.1.2 lis_run_inputs namelist

In the `lis_run_inputs` namelist of the card file these parameters may be reset:

```
LIS%o%EXPCODE
LIS%p%VCLASS
LIS%p%NT
LIS%f%NF
LIS%f%NMIF
LIS%f%ECOR
LIS%o%WFOR
LIS%o%WSINGLE
LIS%o%WPARAM
LIS%o%WTIL
LIS%o%WOUT
LIS%o%STARTCODE
LIS%t%SSS
LIS%t%SMN
LIS%t%SHR
LIS%t%SDA
LIS%t%SMO
```

```

LIS%t%SYR
LIS%t%ENDCODE
LIS%t%ESS
LIS%t%EMN
LIS%t%EHR
LIS%t%EDA
LIS%t%EMO
LIS%t%EYR
LIS%t%TS
LIS%d%UDEF
LIS%o%ODIR
LIS%o%DFILE
LIS%f%GPCPSRC
LIS%f%RADSRC

```

LIS%o%EXPCODE specifies the “experiment code number” for the run. It is used in constructing the name of the output directory for the run. Acceptable values are any 3 digit integer string from 100 through 999.

LIS%p%VCLASS specifies the type of vegetation classification used. The default value is 1 corresponding to the UMD classification.

LIS%p%NT specifies the number of vegetation types. The default value is 13 corresponding to the UMD vegetation type classification.

LIS%f%NF specifies the number of forcing variables. LIS currently uses 10 variables to force the LSMs

LIS%f%NMIF specifies the number of forcing variables for model initialization. The default value is set to 15.

LIS%o%ECOR specifies whether to use elevation correction for forcing.

Acceptable values are:

Value	Description
0	Use elevation correction for forcing
1	Do not use elevation correction for forcing

LIS%o%WFOR specifies whether to output the ALMA optional forcing variables. Acceptable values are:

Value	Description
0	Do not output forcing variables
1	Do output forcing variables

LIS%o%WSINGLE specifies whether to write each variable to a separate file or bundle them together. Acceptable values are:

Value	Description
0	Write all output variables to a single file
1	Write each output variable in a separate file

LIS%o%WPARAM specifies whether to write output for parameters such as the dominant vegetation type, soil type, lai, albedo, gfrac, etc. Acceptable values are:

Value	Description
0	Do not write parameter output
1	Write parameter output

LIS%o%WTIL specifies whether to write output in a tile domain Acceptable values are:

Value	Description
0	Write output in a 2-D grid domain
1	Write output in a 1-D tile domain

LIS%o%WOUT specifies the output data format Acceptable values are:

Value	Description
1	Write output in binary format
2	Write output in grib format

LIS%o%STARTCODE specifies if a restart mode is being used. Acceptable values are:

Value	Description
1	A restart mode is being used
2	A cold start mode is being used, no restart file read

Parameters **LIS%t%SSS**, **LIS%t%SMN**, **LIS%t%SHR**, **LIS%t%SDA**, **LIS%t%SMO**, and **LIS%t%SYR** are used in constructing the starting time for the run. Acceptable values are:²

Variable	Value	Description
LIS%t%SSS	integer 0 – 59	specifying starting second
LIS%t%SMN	integer 0 – 59	specifying starting minute
LIS%t%SHR	integer 0 – 23	specifying starting hour
LIS%t%SDA	integer 1 – 31	specifying starting day
LIS%t%SMO	integer 1 – 12	specifying starting month
LIS%t%SYR	integer 2001 – present	specifying starting year

LIS%o%ENDCODE specifies the termination condition for runs.
Acceptable values are:

Value	Description
0	Terminate the program at real-time date (not currently available)
1	Terminate the program at the specified date

²For this release of the code/data, the start time must be between 01 June 2001 and 30 June 2001

Parameters **LIS%t%ESS**, **LIS%t%EMN**, **LIS%t%EHR**, **LIS%t%EDA**, **LIS%t%EMO**, and **LIS%t%EYR** are used in constructing the ending time for the run. Acceptable values are:³

Variable	Value	Description
LIS%t%ESS	integer 0 – 59	specifying ending second
LIS%t%EMN	integer 0 – 59	specifying ending minute
LIS%t%EHR	integer 0 – 23	specifying ending hour
LIS%t%EDA	integer 1 – 31	specifying ending day
LIS%t%EMO	integer 1 – 12	specifying ending month
LIS%t%EYR	integer 2001 – present	specifying ending year

LIS%t%TS specifies the time-step for the run. Acceptable values are:

Value	Description
900	15 minute time-step
1800	30 minute time-step
3600	60 minute time-step

LIS%o%UDEF specifies the undefined value. The default is set to -9999.

LIS%o%ODIR specifies the name of the top-level output directory. Acceptable values are any 40 character string. The default value of **LIS%o%ODIR** is set to OUTPUT. For simplicity, throughout the rest of this document, this top-level output directory shall be referred to by its default name, *\$WORKING/LIS/OUTPUT*.

LIS%o%DFILE specifies the name of run time diagnostic file. Acceptable values are any 40 character string.

LIS%f%GPCPSRC specifies if an observed precipitation forcing scheme is used or not. Acceptable values are:

Value	Description
0	No observed precipitation scheme used
1	use NRL precipitation product
2	use HUFFMANN precipitation product Currently only CMAP pre-
3	use PERSIAN precipitation product
4	use CMAP precipitation product
5	use CMORPH precipitation product

cipitation product is implemented in LIS. Other schemes are under development.

LIS%f%RADSRC specifies if an observed radiation forcing scheme is used or not. Acceptable values are:

Value	Description
0	No observed radiation scheme used
1	use AGRMET radiation product

³For this release of the code/data, the end time must be between 01 June 2001 and 30 June 2001

8.1.3 domain namelist

The `domain` namelist of the card file specifies the domain information in LIS. LIS uses an array called `kgds` that contains domain definition parameters. This design is adopted from the GRIB grid description used in decoding programs such as `wrf63` [7].

The domain section in the card file defines two types of domain:

- Running domain: defines domain over which simulation is carried out.
- Data domain : defines the domain over which the data is defined.

Currently it is assumed that the forcing data and all parameter data are defined in the same domain. In future releases, the flexibility to define domains for each type of data will be implemented. `kgds` array indices from 0 to 10 defines the running domain, 41 to 50 defines the data domain. The ability to define a running domain different from the data domain enables the user to carry out simulations in only the area of interest. The runtime options for the `kgds` array are shown below. The data domain is defined in a similar fashion with the index starting from 40.

Variable	Description
<code>LIS%d%kgds(1)</code>	0 - Equidistant cylindrical 1 - Mercator cylindrical 3 - Lambert conformal conical 4 - Gaussian cylindrical 5 - Polar stereographic azimuthal
For Latitude/Longitude grids,	
<code>LIS%d%kgds(2)</code>	Number of points on a longitude circle
<code>LIS%d%kgds(3)</code>	Number of points on a latitude circle
<code>LIS%d%kgds(4)</code>	Latitude of origin (x 1000)
<code>LIS%d%kgds(5)</code>	Longitude of origin (x 1000)
<code>LIS%d%kgds(6)</code>	Resolution flag
<code>LIS%d%kgds(7)</code>	Latitude of extreme point (x 1000)
<code>LIS%d%kgds(8)</code>	Longitude of extreme point(x 1000)
<code>LIS%d%kgds(9)</code>	Latitudinal increment
<code>LIS%d%kgds(10)</code>	Longitudinal increment
<code>LIS%d%kgds(11)</code>	Scanning mode flag

For mercator grids,	
LIS%d%kgds(2)	Number of points on a longitude circle
LIS%d%kgds(3)	Number of points on a latitude circle
LIS%d%kgds(4)	Latitude of origin (x 1000)
LIS%d%kgds(5)	Longitude of origin (x 1000)
LIS%d%kgds(6)	Resolution flag
LIS%d%kgds(7)	Latitude of extreme point (x 1000)
LIS%d%kgds(8)	Longitude of extreme point(x 1000)
LIS%d%kgds(9)	Latitude of projection intersection
LIS%d%kgds(10)	Reserved
LIS%d%kgds(11)	Scanning mode flag
LIS%d%kgds(11)	Grid length in longitudinal direction
LIS%d%kgds(11)	Grid length in latitudinal direction
For Lambert Conformal grids,	
LIS%d%kgds(2)	Number of points along X-axis
LIS%d%kgds(3)	Number of points along Y-axis
LIS%d%kgds(4)	Latitude of origin (x 1000)
LIS%d%kgds(5)	Longitude of origin (x 1000)
LIS%d%kgds(6)	Resolution flag
LIS%d%kgds(7)	Lov orientation of grid
LIS%d%kgds(8)	X direction increment
LIS%d%kgds(9)	Y direction increment
LIS%d%kgds(10)	Projection center flag
LIS%d%kgds(11)	Scanning mode flag
For gaussian grids,	
LIS%d%kgds(2)	Number of points on latitude circle
LIS%d%kgds(3)	Number of points on longitude circle
LIS%d%kgds(4)	Latitude of origin (x 1000)
LIS%d%kgds(5)	Longitude of origin (x 1000)
LIS%d%kgds(6)	Resolution flag
LIS%d%kgds(7)	Latitude of extreme point
LIS%d%kgds(8)	Longitude of extreme point
LIS%d%kgds(9)	Longitudinal direction of increment
LIS%d%kgds(10)	Number of circles from poles to equator
LIS%d%kgds(11)	Scanning mode flag
LIS%d%kgds(12)	Number of vertical coordinate parameters
LIS%d%kgds(13)	Octet number of list of vertical coordinate parameters or location of the list of numbers of points in each row or 255 if neither are present

For polar stereographic grids,

LIS%d%kgds(2)	Number of points on a longitude circle
LIS%d%kgds(3)	Number of points on a latitude circle
LIS%d%kgds(4)	Latitude of origin (x 1000)
LIS%d%kgds(5)	Longitude of origin (x 1000)
LIS%d%kgds(6)	Reserved
LIS%d%kgds(7)	Lov grid orientation
LIS%d%kgds(8)	X direction increment
LIS%d%kgds(9)	Y direction increment
LIS%d%kgds(10)	Projection center flag
LIS%d%kgds(11)	Scanning mode flag

Currently LIS supports latitude/longitude and gaussian grids. The support for other grid types are under development.

Parameters LIS%d%MAXT and LIS%d%MINA define the subgrid variability. LIS%d%MAXT defines the maximum tiles per grid (this can be as many as 13, the number of land cover types in the UMD vegetation classification). In addition, users select the smallest percentage of a cell for which to create a tile. LIS%d%MINA defines this parameter. The percentage value is expressed as a fraction.

8.1.4 parms namelist

The **parms** namelist of the card file specifies the names and locations of parameter data common to all land surface models. The data files can be downloaded from the LIS data site at for different domain resolutions. The following parameters can be set using the card file:

```
LIS%p%VFILE  
LIS%p%MFILE  
LIS%p%SAFILE  
LIS%p%CLFILE  
LIS%p%ISCFILE  
LIS%p%ELEVFILE  
LIS%p%AVHRRDIR  
LIS%p%MODISDIR
```

LIS%p%MFILE specifies the location of land/water mask file.

LIS%p%VFILE specifies the location of the vegetation classification file.

LIS%p%SAFILE specifies the sand fraction map file.

LIS%p%CLFILE specifies the clay fraction map file.

LIS%p%ISCFILE specifies the soil color map file.

LIS%p%ELEVFILE specifies the elevation difference between LIS and EDAS (Eta Data Assimilation System) model grids.

LIS%p%AVHRRDIR and LIS%p%MODISDIR specifies the source for reading in LAI/SAI data (real time monthly data or climatology) for AVHRR and MODIS data, respectively. Once the source directory is specified, the program looks for real time data. If the real time data is not available, climatology data is read in.

8.1.5 geos namelist

`geosdrv%GEOSDIR` specifies the location of the GEOS forcing files.

`geosdrv%NCOLD` and `geosdrv%NROLD` specifies the native domain parameters of the GEOS forcing data. The map projection is specified in the driver modules defined for the GEOS routines.

`geosdrv%NMIF` specifies the number of forcing variables provided by GEOS at the model initialization step.

8.1.6 gdas namelist

`gdasdrv%GDASDIR` specifies the location of the GDAS forcing files.

`gdasdrv%NCOLD` and `gdasdrv%NROLD` specifies the native domain parameters of the GDAS forcing data. The map projection is specified in the driver modules defined for the GDAS routines.

`gdasdrv%NMIF` specifies the number of forcing variables provided by GDAS at the model initialization step.

8.1.7 cmap namelist

`cmapdrv%CMAPDIR` specifies the location of the CMAP forcing files.

`cmapdrv%NCOLD` and `cmapdrv%NROLD` specifies the native domain parameters of the CMAP forcing data. The map projection is specified in the driver modules defined for the CMAP routines.

8.1.8 agrmet namelist

`agrmetdrv%AGRMETDIR` specifies the location of the AGRMET forcing files.

8.1.9 clm2 namelist

`clmdrv%WRITEINTC2` defines the output interval for CLM Acceptable values range from 1 to 24. Typical value used in the LIS runs is 3.

`clmdrv%CLM2_RFILE` specifies the CLM active restart file.

`clmdrv%CLM2_CHTFILE` specifies the canopy top and bottom heights for each vegetation type.

`clmdrv%CLM2_ISM` specifies the initial soil moisture used in the cold start runs.

`clmdrv%CLM2_ISCV` specifies the initial snow mass used in the cold start runs.

8.1.10 noah namelist

`noahdrv%WRITEINTN` defines the output interval for Noah Acceptable values range from 1 to 24. Typical value used in the LIS runs is 3.

`noahdrv%NOAH_RFILE` specifies the Noah active restart file.

In the noah namelist of the card file these parameters must correspond with the domain resolution set in the domain namelist:

```
noahdrv%NOAH_MGFILE  
noahdrv%NOAH_ALBFILE
```

`noahdrv%NOAH_MGFILE` and `noahdrv%NOAH_ALBFILE` specify where to find certain Noah related input parameter data files.

`noahdrv%NOAH_VFILE` specifies the Noah static vegetation parameter file.

`noahdrv%NOAH_SFILE` specifies the Noah soil parameter file.

`noahdrv%NOAH_MXSNAL` specifies the Noah max snow free albedo.

`noahdrv%NOAH_TBOT` specifies the Noah bottom temperature

`noahdrv%NOAH_ISM` specifies the initial soil moisture used in the cold start runs.

`noahdrv%NOAH_IT` specifies the initial skin temperature used in the cold start runs.

`noahdrv%NOAH_NVEGP` specifies the number of static vegetation parameters specified for each veg type.

`noahdrv%NOAH_NSOILP` specifies the number of static soil parameters specified.

8.1.11 vic namelist

`vicdrv%WRITEINTV` defines the output interval for VIC Acceptable values range from 1 to 24. Typical value used in LIS runs is 3.

`vicdrv%VIC_NLAYER` specifies the number of soil layers in VIC.

`vicdrv%VIC_NNODE` specifies the number of soil thermal nodes in VIC.

`vicdrv%VIC_NNODE` specifies the number of snow bands in VIC.

`vicdrv%VIC_NNODE` specifies the number of root zones in VIC.

`vicdrv%VIC_SFILE` specifies the VIC soil parameter file.

`vicdrv%VIC_VEGLIBFILE` specifies the VIC vegetation parameter file.

9 Output Data Processing

This section describes how to process the generated output.

The output datasets created by running the LIS executable are written into sub-directories of the `$WORKING/LIS/OUTPUT/` directory (created at run-time). These sub-directories are named either `EXP999` (by default).

The output data consists of ASCII text files and model output in binary format.

For example, assume that you performed a “1/4 deg Noah with GEOS forcing” run, with an experiment code value of 999. This run will produce a `$WORKING/LIS/OUTPUT/EXP999/` directory. This directory will contain:

File Name	Synopsis
Noahstats.dat	Statistical summary of output
lisdiag.dat	Run-time log file (currently empty)
NOAH	Directory containing output data

The `NOAH` directory will contain a `2001/20010611` sub-directory. Its contents are the output files generated by the executable. They are:

LIS.E999.2001061100.NOAHgbin

LIS.E999.2001061103.NOAHgbin

LIS.E999.2001061106.NOAHgbin

LIS.E999.2001061109.NOAHgbin

LIS.E999.2001061112.NOAHgbin

LIS.E999.2001061115.NOAHgbin

LIS.E999.2001061118.NOAHgbin

LIS.E999.2001061121.NOAHgbin

Note, each file-name contains a date-stamp marking the year, month, day, and hour that the data corresponds to. The output data files for CLM and VIC are similar.

The generated output can be written in a 2-D grid format or as a 1-d vector. If written as a 1-d vector, the output must be converted into a 2-d grid before it can be visualized, for example, with GrADS (see <http://grads.iges.org/grads/>). The following section describes some helpful scripts that does this conversion.

9.1 Building `mapto2D`

This sub-section describes how to build the `mapto2D` executable used for converting the 1-d vector data into a 2-d grid.

To build the executable:

1. Change directory into `$WORKING/LIS/postproc`
2. Run the `compile.sh` script: `% sh compile.sh`

9.2 Running `mapto2D`

This sub-section describes how to run the *mapto2D* executable used for converting the 1-d vector data into a 2-d grid.

To post-process the output files run:

1. Change directory into `$WORKING/LIS/postproc`
2. Run *mapto2D*: `% mapto2D file res var lsm`
where:

`file` specifies the name of the particular output file to process.

`res` specifies the resolution of the run. It must be the same value as `LIS%d%DOMAIN`. See Section 8.1.1.

`var` specifies the number of the variable to extract and process. See Tables 9.3, 9.4, and 9.5.

`lsm` specifies the lsm used to generate the output. Acceptable values are `clm`, `noah`, or `vic`.

For example, assume that you performed a “1/4 deg Noah with GEOS forcing” run. To extract and process “Temperature of bare soil” at hour 03 of 11 June 2001:

```
% mapto2D .../OUTPUT/EXP999/NOAH/2001/20010611/LIS.E999.2001061103.NOAHgbin
2 14 noah
```

For this example, *mapto2D* will generate `gridoutV15.1gs4r`.

For those familiar with GrADS, there is a sample GrADS control file, `gridout.ctl`, included in the `$WORKING/LIS/postproc` directory, which may be used to display this example.

9.3 CLM Output

ALMA Mandatory Output

Number	Variable	Description	Units
1	SWnet	Net Shortwave Radiation	W/m ²
2	LWnet	Net Longwave Radiation	W/m ²
3	Qle	Latent Heat Flux	W/m ²
4	Qh	Sensible Heat Flux	W/m ²
5	Qg	Ground Heat Flux	W/m ²
6	Snowf	Snowfall rate	kg/m ² /s
7	Rainf	Rainfall rate	kg/m ² /s
8	Evap	Total Evapotranspiration	kg/m ² /s
9	Qs	Surface Runoff	kg/m ² /s
10	Qsb	Subsurface Runoff	kg/m ² /s
11	Qsm	Snowmelt	kg/m ² /s
12	DelSoilMoist	Change in soil moisture	kg/m ²
13	DelSWE	Change in snow water equivalent	kg/m ²
14	SnowT	Snow Temperature	K
15	VegT	Vegetation Canopy Temperature	K
16	BaresoilT	Temperature of bare soil	K
17	AvgSurfT	Average Surface Temperature	K
18	RadT	Surface Radiative Temperature	K
19	Albedo	Surface Albedo, All Wavelengths	-
20	SWE	Snow Water Equivalent	kg/m ²
21	SoilMoist	Average layer 1 soil moisture	kg/m ²
22	SoilMoist	Average layer 2 soil moisture	kg/m ²
23	SoilMoist	Average layer 3 soil moisture	kg/m ²
24	SoilMoist	Average layer 4 soil moisture	kg/m ²
25	SoilMoist	Average layer 5 soil moisture	kg/m ²

26	SoilMoist	Average layer 6 soil moisture	kg/m ²
27	SoilMoist	Average layer 7 soil moisture	kg/m ²
28	SoilMoist	Average layer 8 soil moisture	kg/m ²
29	SoilMoist	Average layer 9 soil moisture	kg/m ²
30	SoilMoist	Average layer 10 soil moisture	kg/m ²
31	SoilWet	Total Soil Wetness	-
32	TVeg	Vegetation transpiration	kg/m ² /s
33	ESoil	Bare soil evaporation	kg/m ² /s
34	RootMoist	Root zone soil moisture	kg/m ²
35	ACond	Aerodynamic conductance	m/s

ALMA Optional Forcing Output

Number	Variable	Description	Units
36	Wind	Near surface wind magnitude	m/s
37	Rainf	Rainfall rate	kg/m ² /s
38	Snowf	Snowfall rate	kg/m ² /s
39	Tair	Near surface air temperature	K
40	Qair	Near surface specific humidity	kg/kg
41	PSurf	Surface pressure	Pa
42	SWdown	Surface incident shortwave radiation	W/m ²
43	LWdown	Surface incident longwave radiation	W/m ²

9.4 Noah Output

ALMA Mandatory Output

Number	Variable	Description	Units
1	SWnet	Net Shortwave Radiation	W/m ²
2	LWnet	Net Longwave Radiation	W/m ²
3	Qle	Latent Heat Flux	W/m ²
4	Qh	Sensible Heat Flux	W/m ²
5	Qg	Ground Heat Flux	W/m ²
6	Snowf	Snowfall rate	kg/m ² /s
7	Rainf	Rainfall rate	kg/m ² /s
8	Evap	Total Evapotranspiration	kg/m ² /s
9	Qs	Surface Runoff	kg/m ² /s
10	Qsb	Subsurface Runoff	kg/m ² /s
11	Qsm	Snowmelt	kg/m ² /s
12	DelSoilMoist	Change in soil moisture	kg/m ²
13	DelSWE	Change in snow water equivalent	kg/m ²
14	AvgSurfT	Average Surface Temperature	K
15	Albedo	Surface Albedo, All Wavelengths	-
16	SWE	Snow Water Equivalent	kg/m ²
17	SoilMoist	Average layer 1 soil moisture	kg/m ²
18	SoilMoist	Average layer 2 soil moisture	kg/m ²
19	SoilMoist	Average layer 3 soil moisture	kg/m ²
20	SoilMoist	Average layer 4 soil moisture	kg/m ²
21	SoilWet	Total Soil Wetness	-
22	TVeg	Vegetation transpiration	kg/m ² /s
23	ESoil	Bare soil evaporation	kg/m ² /s
24	RootMoist	Root zone soil moisture	kg/m ²

ALMA Optional Forcing Output

Number	Variable	Description	Units
25	Wind	Near surface wind magnitude	m/s
26	Rainf	Rainfall rate	kg/m ² /s
37	Snowf	Snowfall rate	kg/m ² /s
28	Tair	Near surface air temperature	K
29	Qair	Near surface specific humidity	kg/kg
30	PSurf	Surface pressure	Pa
31	SWdown	Surface incident shortwave radiation	W/m ²
32	LWdown	Surface incident longwave radiation	W/m ²

9.5 VIC Output

ALMA Mandatory Output

Number	Variable	Description	Units
1	SWnet	Net Shortwave Radiation	W/m ²
2	LWnet	Net Longwave Radiation	W/m ²
3	Qle	Latent Heat Flux	W/m ²
4	Qh	Sensible Heat Flux	W/m ²
5	Qg	Ground Heat Flux	W/m ²
6	Rainf	Rainfall rate	kg/m ² /s
7	Snowf	Snowfall rate	kg/m ² /s
8	Evap	Total Evapotranspiration	kg/m ² /s
9	Qs	Surface Runoff	kg/m ² /s
10	Qsb	Subsurface Runoff	kg/m ² /s
11	Qfz	Re-freezing of water in the snow	kg/m ² /s
12	SnowT	Snow Temperature	K
13	AvgSurfT	Average Surface Temperature	K
14	RadT	Surface Radiative Temperature	K
15	Albedo	Surface Albedo, All Wavelengths	-
16	SoilMoist	Average layer 1 soil moisture	kg/m ²
17	SoilMoist	Average layer 2 soil moisture	kg/m ²
18	SoilMoist	Average layer 3 soil moisture	kg/m ²
19	TVeg	Vegetation transpiration	kg/m ² /s
20	ESoil	Bare soil evaporation	kg/m ² /s
21	SoilWet	Total Soil Wetness	-
22	RootMoist	Root zone soil moisture	kg/m ²
23	SWE	Snow Water Equivalent	kg/m ²
24	Qsm	Snowmelt	kg/m ² /s
25	DelSoilMoist	Change in soil moisture	kg/m ²
26	DelSWE	Change in snow water equivalent	kg/m ²
27	ACond	Aerodynamic conductance	m/s

ALMA Optional Forcing Output

Number	Variable	Description	Units
28	Wind	Near surface wind magnitude	m/s
29	Tair	Near surface air temperature	K
30	PSurf	Surface pressure	Pa
31	SWdown	Surface incident shortwave radiation	W/m ²
32	LWdown	Surface incident longwave radiation	W/m ²

A LIS Card File

```
&driver
LIS%d%DOMAIN = 4
LIS%d%LSM = 1
LIS%f%FORCE = 2
LIS%d%SOIL = 2
LIS%p%LAI = 2
/

&lis_run_inputs
LIS%o%EXPCODE = 444
LIS%p%VCLASS = 1
LIS%p%NT = 13
LIS%f%NF = 10
LIS%f%NMIF = 15
LIS%f%ECOR = 0
LIS%o%WFOR = 1
LIS%o%WSINGLE = 0
LIS%o%WPARAM = 0
LIS%o%WTIL = 0
LIS%o%WOUT = 1
LIS%o%STARTCODE = 2
LIS%t%SSS = 0
LIS%t%SMN = 00
LIS%t%SHR = 21
LIS%t%SDA = 10
LIS%t%SMO = 05
LIS%t%SYR = 2001
LIS%t%ENDCODE = 1
LIS%t%ESS = 0
LIS%t%EMN = 00
LIS%t%EHR = 21
LIS%t%EDA = 11
LIS%t%EMO = 05
LIS%t%EYR = 2001
LIS%t%TS = 1800
LIS%d%UDEF = -9999.
LIS%o%ODIR = "OUTPUT"
LIS%o%DFILE = "lisdiag"
LIS%f%GPCPSRC = 0
LIS%f%RADSRC = 0
/
&domain
LIS%d%kgds(1) = 0
LIS%d%kgds(2) = 20
```

```

LIS%d%kgds(3)      = 17
LIS%d%kgds(4)      = 27875
LIS%d%kgds(5)      = -97625
LIS%d%kgds(6)      = 128
LIS%d%kgds(7)      = 31875
LIS%d%kgds(8)      = -92875
LIS%d%kgds(9)      = 250
LIS%d%kgds(10)     = 250
LIS%d%kgds(11)     = 64
LIS%d%kgds(20)     = 255
LIS%d%kgds(41)     = 0
LIS%d%kgds(42)     = 1440
LIS%d%kgds(43)     = 600
LIS%d%kgds(44)     = -59875
LIS%d%kgds(45)     = -179875
LIS%d%kgds(46)     = 128
LIS%d%kgds(47)     = 89875
LIS%d%kgds(48)     = 179875
LIS%d%kgds(49)     = 250
LIS%d%kgds(50)     = 250
LIS%d%MAXT          = 1
LIS%d%MINA          = 0.05
/
&parms
LIS%p%MFILE         = "GVEG/1_4deg/UMD_AVHRR60mask0.25.bfsa"
LIS%p%VFILE         = "GVEG/1_4deg/UMD_AVHRR60G0.25.bfsa"
LIS%p%SAFILE        = "BCS/1_4deg/sand60.bfsa"
LIS%p%CLFILE        = "BCS/1_4deg/clay60.bfsa"
LIS%p%ISCFFILE      = "BCS/1_4deg/soicol60.bfsa"
LIS%p%ELEVFILE      = "GVEG/1_4deg/eldif_geos3_60.25.bin"
LIS%p%AVHRRDIR      = "./input/AVHRR_LAI"
LIS%p%MODISDIR      = "./input/MODIS_LAI"
/
&geos
geosdrv%GEOSDIR    = "/X6RAID/DATA/GEOS/BEST_LK"
geosdrv%NROLD       = 181
geosdrv%NCOLD       = 360
geosdrv%NMIF        = 13
/
&gdas
gdasdrv%GDASDIR   = "./input/FORCING/GDAS"
gdasdrv%NROLD       = 256
gdasdrv%NCOLD       = 512
gdasdrv%NMIF        = 15
/

```

```

&cmap
cmapdrv%CMAPDIR      = "./input/CMAP"
cmapdrv%NROLD        = 181
cmapdrv%NCOLD         = 360
/
&agrmet
agrmetdrv%AGRMETDIR  = "/X6RAID/DATA/AGRMET"
/
&clm2
clmdrv%WRITEINTC2    = 3
clmdrv%CLM2_RFILE    = "clm2.rst"
clmdrv%CLM2_VFILE    = "BCS/clm_parms/umdvegparam.txt"
clmdrv%CLM2_CHTFILE  = "BCS/clm_parms/clm2_ptcanhts.txt"
clmdrv%CLM2_ISM       = 0.45
clmdrv%CLM2_IT        = 290.0
clmdrv%CLM2_ISCV     = 0.
/
&noah
noahdrv%WRITEINTN    = 3
noahdrv%NOAH_RFILE   = "noah.rst"
noahdrv%NOAH_MGFILE  = "BCS/1_4deg/NOAH/"
noahdrv%NOAH_ALBFILE = "BCS/1_4deg/NOAH/"
noahdrv%NOAH_VFILE   = "BCS/noah_parms/noah.vegparms.txt"
noahdrv%NOAH_SFILE   = "BCS/noah_parms/noah.soilparms.txt"
noahdrv%NOAH_MXSNAL  = "BCS/1_4deg/NOAH/maxsnalb.bfsa"
noahdrv%NOAH_TBOT    = "BCS/1_4deg/NOAH/tbot.bfsa"
noahdrv%NOAH_ISM     = 0.30
noahdrv%NOAH_IT      = 290.0
noahdrv%NOAH_NVEGP   = 7
noahdrv%NOAH_NSOILP  = 10
/
&vic
vicdrv%WRITEINTVIC   = 3
vicdrv%VIC_NLAYER    = 3
vicdrv%VIC_NNODE      = 5
vicdrv%VIC_SNOWBAND   = 1
vicdrv%VIC_ROOTZONES = 2
vicdrv%VIC_SFILE      = "./BCS/vic_parms/soil.txt"
vicdrv%VIC_VEGLIBFILE = "./BCS/vic_parms/veg_lib.txt"
/
&mos
mosdrv%WRITEINTM     = 3
mosdrv%MOS_RFILE     = "mos.rst"
mosdrv%MOS_MFILE     = "mosgdas.rst"
mosdrv%MOS_VFILE     = "./BCS/mos_parms/real.vegiparms.txt"

```

```
mosdrv%MOS_MVFILE      = "./BCS/mos_parms/real.monvegpar.txt"
mosdrv%MOS_SFILE        = "./BCS/mos_parms/real.soilparms.txt"
mosdrv%MOS_KVFILE        = "./BCS/mos_parms/real.vegiparms.randy.txt"
mosdrv%MOS_KMVFIL        = "./BCS/mos_parms/real.monvegpar.randy.txt"
mosdrv%MOS_KSFILE        = "./BCS/mos_parms/real.soilparms.randy.txt"
mosdrv%MOS_POFILE        =
mosdrv%MOS_SIFILE        =
mosdrv%MOS_SLFILE        =
mosdrv%MOS_ISM           = 0.3
mosdrv%MOS_IT            = 290.0
mosdrv%MOS_IC            = 1
mosdrv%MOS_SMDA          = 0
mosdrv%MOS_TDA           = 0
mosdrv%MOS_SDA           = 0
mosdrv%MOS_NVEGP         = 24
mosdrv%MOS_NMVEGP        = 6
mosdrv%MOS_NSOILP        = 10
mosdrv%MOS_NRET          = 64
/

```

B Makefile

```
# Set up special characters
null  :=
space := $(null) $(null)
doctool :=../../utils/docsgen.sh

# Check for directory in which to put executable
ifeq ($(MODEL_EXEDIR),$(null))
MODEL_EXEDIR := .
endif

# Check for name of executable
ifeq ($(EXENAME),$(null))
EXENAME := LIS
endif

# Check if SPMD is defined in "misc.h"
# Ensure that it is defined and not just "undef SPMD" set in file
ifeq ($(SPMD),$(null))
    SPMDSSET := $(shell /bin/grep SPMD misc.h)
    ifneq (,$(findstring define,$(SPMDSET)))
        SPMD := TRUE
    else
        SPMD := FALSE
    endif
endif

# Determine platform
UNAMES := $(shell uname -s)
UMACHINE := $(shell uname -m)

ifeq ($(UNAMES),IRIX64)
ESMF_DIR    := ../../lib/esmf
LIB_ESMF    := $(ESMF_DIR)/lib/lib0
MOD_ESMF    := $(ESMF_DIR)/mod/mod0

endif

ifeq ($(UNAMES),OSF1)

LIB_MPI := /usr/lib
INC_MPI := /usr/include
ESMF_DIR    := ../../lib/esmf
LIB_ESMF    := $(ESMF_DIR)/lib/lib0
MOD_ESMF    := $(ESMF_DIR)/mod/mod0
```

```

endif

ifeq ($(UMACHINE), i686)

MPI_PREFIX := /data1/jim/local/mpich-1.2.4-absoft
LIB_MPI := $(MPI_PREFIX)/lib
INC_MPI := $(MPI_PREFIX)/include
ESMF_DIR := ./lib/esmf
LIB_ESMF := $(ESMF_DIR)/lib/lib0
MOD_ESMF := $(ESMF_DIR)/mod/mod0

endif

# Load dependency search path.
dirs := . $(shell cat Filepath)
# Set cpp search path, include netcdf
cpp_dirs := $(dirs) $(INC_NETCDF) $(INC_MPI)
cpp_path := $(foreach dir,$(cpp_dirs),-I$(dir)) # format for command line

# Expand any tildes in directory names. Change spaces to colons.
VPATH := $(foreach dir,$(cpp_dirs),$(wildcard $(dir)))
VPATH := $(subst $(_space),:,$(VPATH))

#-----
# Primary target: build the model
#-----
all: $(MODEL_EXEDIR)/$(EXENAME)

# Get list of files and determine objects and dependency files
FIND_FILES = $(wildcard $(dir)/*.F $(dir)/*.f $(dir)/*.F90 $(dir)/*.c)
FILES      = $(foreach dir, $(dirs),$(FIND_FILES))
SOURCES    := $(sort $(notdir $(FILES)))
DEPS       := $(addsuffix .d, $(basename $(SOURCES)))
OBJS       := $(addsuffix .o, $(basename $(SOURCES)))
DOCS       := $(addsuffix .tex, $(basename $(SOURCES)))

$(MODEL_EXEDIR)/$(EXENAME): $(OBJS)
$(FC) -o $@ $(OBJS) $(FOPTS) $(LDFLAGS)
debug: $(OBJS)
    echo "FFLAGS: $(FFLAGS)"
    echo "LDFLAGS: $(LDFLAGS)"
    echo "OBJS: $(OBJS)"

*****
***** Architecture-specific flags and rules *****

```

```

*****  

#-----  

# SGI  

#-----  

  

ifeq ($(UNAMES),IRIX64)  

  

ESMF_ARCH = IRIX64  

FC        := f90  

CPP       := /lib/cpp  

  

# Library directories  

LIB_DIR  = ../lib/sgi-64/  

HDFLIBDIR = $(LIB_DIR)  

GFIOLIBDIR = $(LIB_DIR)  

CPPFLAGS   := -P  

PSASINC    :=  

CFLAGS      := $(cpp_path) -64 -c -O2 -fno-strict-aliasing -fno-common -fno-�  

FFLAGS      = $(cpp_path) -64 -r4 -i4 -c -fno-strict-aliasing -fno-common -fno-�  

-DHIDE_SHR_MSG -DNO_SHR_VMATH -DIRIX64 -O2 -fno-strict-aliasing -fno-common -fno-�  

# Use this option if using OPENDAP mode  

#FFLAGS      = $(cpp_path) -64 -r4 -i4 -c -fno-strict-aliasing -fno-common -fno-�  

-DHIDE_SHR_MSG -DNO_SHR_VMATH -DIRIX64 -O2 -fno-strict-aliasing -fno-common -fno-�  

FOPTS = $(LIB_DIR)bacio_64_sgi $(LIB_DIR)w3lib_64_sgi  

LDFLAGS     = -64 -L$(LIB_ESMF)/$(ESMF_ARCH) -lesmf -lmpi  

# WARNING: -mp and -g together cause wrong answers  

  

# WARNING: - Don't run hybrid on SGI (that's what the == -mp is all about)  

  

ifeq ($(SPMD),TRUE)  

FFLAGS  -= -mp  

FFLAGS  += -macro_expand  

# FFLAGS  += -I$(INC_MPI) -macro_expand  

  

# LDFLAGS += -L$(LIB_MPI) -lmpi  

else  

FFLAGS  += -DHIDE_MPI  

endif  

  

.SUFFIXES:  

.SUFFIXES: .F90 .c .o  

  

.F90.o:  

$(FC) $(FFLAGS) $<

```

```

.c.o:
cc $(cpp_path) $(CFLAGS) $<

endif
#-----
# Compaq alpha - Halem cluster
#-----
ifeq ($(UNAMES),OSF1)

ESMF_ARCH = alpha
FC        := f90
CPP       := /lib/cpp

# Library directories
LIB_DIR  = ./lib/alpha-32/
CPPFLAGS := -P
PSASINC  :=
CFLAGS   := $(cpp_path) -n32 -DOSF1
FFLAGS   = $(cpp_path) -c -cpp -automatic -convert big_endian
-assume byterecl -arch ev6 -tune ev6 -fpe3\
-I$(MOD_ESMF)/$(ESMF_ARCH) -DOSF1 \
-DHIDE_SHR_MSG -DNO_SHR_VMATH
FFLAGS_DOTF90 = -DHIDE_SHR_MSG -DOSF1 -free -fpe3 -DNO_SHR_VMATH
FFLAGS_DOTF = -extend_source -omp -automatic
FOPTS = $(LIB_DIR)bacio_32_alpha $(LIB_DIR)w3lib_32_alpha
LDFLAGS   = -L$(LIB_ESMF)/$(ESMF_ARCH) -lesmf -lmpi
# WARNING: -mp and -g together cause wrong answers

# WARNING: - Don't run hybrid on SGI (that's what the == -mp is all about)

ifeq ($(SPMD),TRUE)
FFLAGS == -mp
FFLAGS += -I$(INC_MPI) -macro_expand

# LDFLAGS += -L$(LIB_MPI) -lmpi
else
FFLAGS += -DHIDE_MPI
endif

.SUFFIXES:
.SUFFIXES: .f .f90 .F90 .c .o

.f.o:
$(FC) $(FFLAGS) $<
.F90.o:
$(FC) $(FFLAGS) $<

```

```

.c.o:
cc -c $(cpp_path) $(CFLAGS) $<
.f90.o:
$(FC) $(FFLAGS) $<

endif
#-----
# Linux
#-----
-----
```

```

ifeq ($(UMACHINE),i686)
ESMF_ARCH = linux_absoft

ifeq ($(ESMF_ARCH),linux_ifc)

FC      := $(MPI_PREFIX)/bin/mpif90
CPP     := /lib/cpp

CFLAGS   := $(cpp_path) -c -O2
FFLAGS   = $(cpp_path) -c -I$(MOD_ESMF)/$(ESMF_ARCH) -DHIDE_SHR_MSG
-DNO_SHR_VMATH -O
LDFLAGS  = -L$(LIB_ESMF)/$(ESMF_ARCH) -lesmf -lmpich

endif

ifeq ($(ESMF_ARCH),linux_absoft)

FC      := $(MPI_PREFIX)/bin/mpif90
CC      := $(MPI_PREFIX)/bin/mpicc
CPP     := /lib/cpp

CFLAGS   := $(cpp_path) -c -O2 -DABSOFT -DLITTLE_ENDIAN
FFLAGS   = $(cpp_path) -c -O2 -YEXT_NAMES=LCS -s -B108 -YCFRL=1
-p$(MOD_ESMF)/$(ESMF_ARCH) -DHIDE_SHR_MSG -DNO_SHR_VMATH -DABSOFT
-DLITTLE_ENDIAN
# Use this option if using OPENDAP mode
#CFLAGS    := $(cpp_path) -c -O2 -DABSOFT -DLITTLE_ENDIAN -DOPENDAP
#FFLAGS    = $(cpp_path) -c -O2 -YEXT_NAMES=LCS -s -B108 -YCFRL=1
-p$(MOD_ESMF)/$(ESMF_ARCH) -DHIDE_SHR_MSG -DNO_SHR_VMATH -DABSOFT
-DLITTLE_ENDIAN -DOPENDAP
LDFLAGS  = -L$(LIB_ESMF)/$(ESMF_ARCH) -lesmf -lmpich -lU77 -lm

endif

ifeq ($(ESMF_ARCH),linux_lf95)
```

```

FC      := $(MPI_PREFIX)/bin/mpif90
CPP    := /lib/cpp
CFLAGS := $(cpp_path) -c -O -DUSE_GCC -DLAHEY -DLITTLE_ENDIAN
FFLAGS := $(cpp_path) -c -O -DHIDE_SHR_MSG -DLINUX -DNO_SHR_VMATH
-I$(MOD_ESMF)/$(ESMF_ARCH) -DLAHEY -DLITTLE_ENDIAN
LDFLAGS := -L$(LIB_ESMF)/$(ESMF_ARCH) -lesmf -L$(LIB_MPI) -lmpich
-s --staticlink

endif

# Library directories
LIB_DIR  = ./lib/pc-32/$(ESMF_ARCH)/
HDFLIBDIR = $(LIB_DIR)
GFIOLIBDIR = $(LIB_DIR)
CPPFLAGS   := -P
PSASINC    :=
FOPTS = $(LIB_DIR)bacio_32_pclinux $(LIB_DIR)w3lib_32_pclinux
# WARNING: -mp and -g together cause wrong answers

# WARNING: - Don't run hybrid on SGI (that's what the -= -mp is all about)

ifeq ($(SPMD),TRUE)
# FFLAGS -= -mp
# FFLAGS += -macro_expand
# FFLAGS += -I$(INC_MPI) -macro_expand

# LDFLAGS += -L$(LIB_MPI) -lmpi
else
  FFLAGS += -DHIDE_MPI
endif

.SUFFIXES:
.SUFFIXES: .F90 .c .o

.F90.o:
$(FC) $(FFLAGS) $<
.c.o:
$(CC) $(cpp_path) $(CFLAGS) $<

endif

RM := rm
# Add user defined compiler flags if set, and replace FC if USER option set.
FFLAGS += $(USER_FFLAGS)

```

```

ifeq ($(USER_FC),$(null))
FC := $(USER_FC)
endif

clean:
$(RM) -f *.o *.mod *.stb $(MODEL_EXEDIR)/$(EXENAME)

realclean:
$(RM) -f *.o *.d *.mod *.stb $(MODEL_EXEDIR)/$(EXENAME)
doc:
$(doctool)
#-----
#!!!!!!!!!!!!!!DO NOT EDIT BELOW THIS LINE.!!!!!!!!!!!!!!
#-----
# These rules cause a dependency file to be generated for each source
# file. It is assumed that the tool "makdep" (provided with this
# distribution in clm2/tools/makdep) has been built and is available in
# the user's $PATH. Files contained in the clm2 distribution are the
# only files which are considered in generating each dependency. The
# following filters are applied to exclude any files which are not in
# the distribution (e.g. system header files like stdio.h).
#
# 1) Remove full paths from dependencies. This means gnumake will not break
#    if new versions of files are created in the directory hierarchy
#    specified by VPATH.
#
# 2) Because of 1) above, remove any file dependencies for files not in the
#    clm2 source distribution.
#
# Finally, add the dependency file as a target of the dependency rules. This
# is done so that the dependency file will automatically be regenerated
# when necessary.
#
#      i.e. change rule
#          make.o : make.c make.h
#          to:
#          make.o make.d : make.c make.h
#-----
DEPGEN := ./MAKDEP/makdep -s F
%.d : %.c
@echo "Building dependency file $@"
@$(DEPGEN) -f $(cpp_path) $< > $@
%.d : %.f
@echo "Building dependency file $@"
@$(DEPGEN) -f $(cpp_path) $< > $@
%.d : %.F90

```

```

@echo "Building dependency file $@"
@$(DEPGEN) -f $(cpp_path) $< > $@
%.d : %.F
@echo "Building dependency file $@"
@$(DEPGEN) -f $(cpp_path) $< > $@
#
# if goal is clean or realclean then don't include .d files
# without this is a hack, missing dependency files will be created
# and then deleted as part of the cleaning process
#
INCLUDE_DEPS=TRUE
ifeq ($(MAKECMDGOALS), realclean)
    INCLUDE_DEPS=FALSE
endif
ifeq ($(MAKECMDGOALS), clean)
    INCLUDE_DEPS=FALSE
endif

ifeq ($(INCLUDE_DEPS), TRUE)
    -include $(DEPS)
endif

```

References

- [1] GrADS. <http://grads.iges.org/grads/grads.html>.
- [2] Protex documenting system. <http://gmao.gsfc.nasa.gov/software/protex>.
- [3] CLM. <http://www.cgd.ucar.edu/tss/clm>.
- [4] DODS. <http://www.unidata.ucar.edu/packages/dods/>.
- [5] NOAH. <ftp://ftp.ncep.noaa.gov/pub/gcp/lidas/noahlsm/>.
- [6] VIC. <http://hydrology.princeton.edu/research/lis/index.html>.
- [7] W3FI63 PROGRAM. <http://dss.ucar.edu/datasets/ds609.1/software/mords/w3fi63.f>.
- [8] G. J. Collatz, C Grivet, J. T. Ball, and J. A. Berry. Physiological and environmental regulation of stomatal conductance: Photosynthesis and transpiration: A model that includes a laminar boundary layer. *Agric. For. Meteorol.*, 5:107–136, 1991.
- [9] Chen. F., Mitchell. K., Schaake. J, Xue. J, Pan. H, Koren. V., Ek. M Duan, and A. Betts. Modeling of land-surface evaporation by four schemes and comparison with fife observations. *J. Geophys. Res.*, 101(D3):7251–7268, 1996.
- [10] P. G. Jarvis. The interpretation of leaf water potential and stomatal conductance found in canopies of the field. *Phil. Trans. R. Soc.*, 273:593–610, 1976.
- [11] L. A. Richards. Capillary conduction of liquids in porous media. *Physics*, 1:318–333, 1931.
- [12] E. Rogers, T. L. Black, D. G. Deaven, G. J. DiMego, Q. Zhao, M. Baldwin, N. W. Junker, and Y. Lin. Changes to the operational "early" eta analysis/forecast system at the national centers of environmental prediction. *Wea. Forecasting*, 11:391–413, 1996.